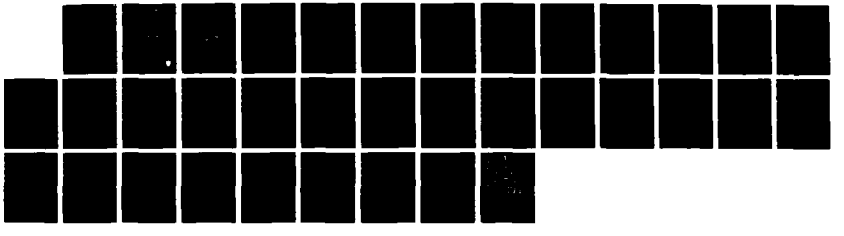
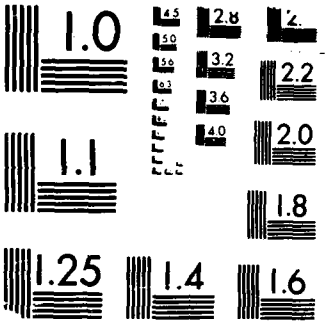
AD-A194 032 FASTER SCALING ALGORITHMS FOR NETWORK PROBLEMS(U) 1/1
PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE
H N GABOW ET AL. AUG 87 CS-TR-111-87 N00014-87-K-0467
UNCLASSIFIED F/G 12/4 NL





(4)

Princeton University

FASTER SCALING ALGORITHMS FOR NETWORK PROBLEMS

Harold N. Gabow
Robert E. Tarjan

CS-TR-111-87

August 1987

Department of Computer Science

(4)

FASTER SCALING ALGORITHMS FOR NETWORK PROBLEMS

Harold N. Gabow
Robert E. Tarjan

CS-TR-111-87

August 1987

Faster Scaling Algorithms for Network Problems

Harold N. Gabow[1]
Department of Computer Science
University of Colorado
Boulder, CO
80309

Robert E. Tarjan[2]
Computer Science Department
Princeton University
Princeton, NJ 08544
and
AT&T Bell Laboratories
Murray Hill, NJ 07974

August 8, 1987

Abstract.

This paper presents algorithms for the assignment problem, the transportation problem and the minimum cost flow problem of operations research. The algorithms find a minimum cost solution, but run in time close to the best-known bounds for the corresponding problems without costs. For example, the assignment problem (equivalently, minimum cost matching on a bipartite graph) can be solved in $O(\sqrt{n}m\log(nN))$ time, where n, m and N denote the number of vertices, number of edges and largest magnitude of a cost; costs are assumed to be integral. The algorithms work by scaling. As in the work of Goldberg and Tarjan [Go, GoT87a-b], in each scaled problem an approximate optimum solution is found, rather than an exact optimum.

---

[1] Research supported in part by NSF Grant No. MCS-8302648 and AT&T Bell Laboratories.

[2] Research supported in part by NSF Grant No. DCR-8605962 and ONR Contract No. N00014-87-K-0467.

1. Introduction.

Many problems in operations research involve minimizing a cost function defined on a bipartite or directed graph. A simple but fundamental example is the assignment problem. This paper gives algorithms for such problems that run almost as fast as the best-known algorithms for the corresponding problems without costs. For the assignment problem, the corresponding problem without costs is maximum cardinality bipartite matching.

The results are achieved by scaling the costs. This requires the costs to be integral-valued. Further, for the algorithms to be efficient, costs should be polynomially-bounded in the number of vertices, i.e., $n^{O(1)}$. These requirements are satisfied by a large number of problems in both theoretical and practical applications.

Table I summarizes the results of the paper. The parameters describing the input are specified in the caption and defined more precisely below. The first column gives the problem and the best-known strong-polynomial time bound. Such a bound comes from an algorithm with run time independent of the size of the numbers (assuming the uniform cost model of computation [AHU]). The second column gives the time bounds achieved in this paper by scaling. The table shows that significant speed-ups can be achieved through scaling. Further it will be seen that the scaling algorithms are simple to program. Now we discuss the specific results.

The assignment problem is to find a minimum cost perfect matching on a bipartite graph. The strong polynomial algorithm is the Hungarian algorithm [K55,K56] implemented with Fibonacci heaps [FT]. This algorithm can be significantly improved when all costs are zero. Then the problem amounts to finding a perfect matching. The best-known cardinality matching algorithm, due to Hopcroft and Karp, runs in time $O(\sqrt{n}m)$ [HK]. The new time bound for the assignment problem is just a factor $\log nN$ more than this. The algorithm is similar to the Hopcroft-Karp cardinality algorithm and appears simple enough to be useful in practice.

The new algorithm improves the scaling algorithm of [G85a], which runs in time $O(n^{3/4}m \log N)$. The improvement comes from a different scaling method. The algorithms of [G85a] compute an optimum solution at each of $\log N$ scales. The new method computes an approximate optimum at each of $\log nN$ scales; using $\log n$ extra scales ensures that the last approximate optimum is exact. The appropriate definition of approximate optimum is due to Bertsekas [Ber86]. The new approach to scaling was recently discovered by Goldberg and Tarjan for the minimum cost flow problem [Go, GoT87a-b]. Their minimum cost flow algorithm solves the assignment problem in time $O(nm \log(nN))$, which this paper improves. Bertsekas [Ber87] gives an algorithm for the

assignment problem that also runs sequential time $O(nm \log(nN))$, and in practice appears to run faster on parallel machines.

The assignment algorithm extends to other network problems. Variants of minimum cost perfect matching (such as minimum cost matching) can be done in the same time bound. The linear programming dual variables for perfect matching can be found from the algorithm. This gives a solution to the shortest path problem when negative edge lengths are allowed. The table entry for the degree-constrained subgraph problem is just a factor $\log nN$ more than the bound of [ET] for the corresponding problem without costs, namely the problem of maximum flow on a 0-1 network. These bounds improve [G85a] in a manner analogous to matching.

The table entry for the transportation problem is a good bound when total supply and demand (U) is small. The key fact for this bound is the low total augmenting path length for the assignment algorithm; this fact generalizes the bounds of [ET] for cardinality matching and 0-1 network flow. The entry for minimum cost flow is a double scaling algorithm— it scales edge capacities, and at each scale solves a small transportation problem by the above cost-scaling algorithm. This algorithm is not as good asymptotically as the recent bound of Goldberg and Tarjan, $O(nm \log(n^2/m) \log(nN))$ [GoT87b]. The latter is just a factor $\log nN$ more than the best bound for maximum value flow [GoT86]. However the double scaling algorithm may be more useful in practice, since it requires fewer data structures. Furthermore it generalizes to allow the cost of an edge to be an arbitrary convex function of the flow in the edge; the time bound is unchanged, as long as the cost for a given flow value can be computed in $O(1)$ time. The last table entry, for the Chinese postman problem, is a consequence of the mincost flow algorithm. The algorithm scales capacities but the bound is strongly polynomial.

Section 2 presents the matching algorithm and its analysis, including facts used in the generalizations. Section 3 presents the extensions to more general network flow problems. Section 4 gives brief concluding remarks. This section closes with definitions from graph theory; more thorough treatments are in [L, T83].

We use interval notation for sets of integers. Thus for integers i and j, $[i..j] = \{k | k \text{ is an integer, } i \leq j \leq k\}$, $[i..j) = \{k | k \text{ is an integer, } i \leq j < k\}$, etc. The function *log* n denotes logarithm to the base two.

For a graph G, $V(G)$ and $E(G)$ denote the vertex set and edge set, respectively. The given graph G is bipartite and has bipartition V_0, V_1 (so $V(G)$ partitions into V_0 and V_1, and any edge joins V_0 to V_1). The given graph G has m edges; in Section 2, $n = |V_0| = |V_1|$ (we assume without loss of generality that the two sets of the bipartition have equal cardinality); in Section 3,

$n = |V(G)|$. If H is a subgraph of G, an *H-edge* is an edge in H and a *non-H-edge* is not in H. When an auxilary graph G' is constructed from the given graph G, *G-edge* refers to an edge of G' that represents an edge of G. We use this term without explicit comment only when the representation is obvious ($vw \in E(G)$ is represented by $v'w'$ where v' and w' are obvious representatives of v and w). We say *path P ends with edge vw* if vw is at an end of P and further, v is an endpoint of P.

A *matching* on a graph is a set of vertex-disjoint edges. Thus a vertex v is in at most one matched edge vv'; v' is the *mate* of v. A *free* vertex has no mate. A *maximum cardinality matching* has the greatest number of edges possible; a *perfect matching* has no free vertices (and is clearly maximum cardinality). An *alternating path (cycle)* for a matching is a simple path (cycle) whose edges are alternately matched and unmatched. An *augmenting path P* is an alternating path joining two distinct free vertices. *Augmenting* the matching along P means enlarging the matching M to $M \oplus P$, thus giving a matching with one more edge. Suppose each edge e has a numeric *cost* $c(e)$; in this paper costs are integers in $[-N..N]$ unless stated otherwise. The cost $c(S)$ of a set of edges S is the sum of the individual edge costs. A *minimum* (*maximum*) *perfect matching* is a perfect matching of smallest (largest) possible cost. The *assignment problem* is to find a minimum perfect matching on a bipartite graph. More generally, a *minimum cost maximum cardinality matching* is a matching that has the greatest number of edges possible, and subject to that restriction has minimum cost possible. (The phrase "minimum cost maximum cardinality set" can be interpreted ambiguously. In this paper it refers to a set that has maximum cardinality subject to any other restrictions that have been mentioned, and among such sets has minimum cost possible). A *minimum cost matching* is a matching of minimum cost (its cardinality can be any value, including zero).

A *multigraph* has a set of edges $E(G)$, where each edge e has an integral *multiplicity* $u(e)$ (i.e., there are $u(e)$ parallel copies of e). The size parameter m is the number of edges, $m = |E(G)|$; $\overline{m}$ counts multiplicities, i.e., $\overline{m} = \sum\{u(e) | e \in E(G)\}$; M is the maximum edge multiplicity. (In a graph $M = 1$). When each vertex v has associated nonnegative integers $\ell(v)$ and $u(v)$, a *degree-constrained subgraph* (*DCS*) is a subgraph where each vertex has degree in $[\ell(v)..u(v)]$. It is convenient to use both set notation and functional notation for a DCS. Thus we use a capital letter D to denote a DCS, and the corresponding lower case letter d to denote two functions defined by D: for an edge e, $d(e)$ denotes the multiplicity of e in D, and for a vertex v, $d(v)$ denotes the degree of v in D, i.e., $d(v) = \sum\{d(vw) | vw \in E(G)\}$. Hence $d(e) \leq u(e)$ and $\ell(v) \leq d(e) \leq u(v)$. The *deficiency* of DCS D at vertex v is $\phi(v, D) = u(v) - d(v)$. In a *perfect DCS* each deficiency is zero. The size of the DCS is measured by $U = \sum\{u(v) | v \in V\}$ (so U is twice the number of edges in a perfect DCS). When edges e have costs the usual assumption is that each copy of e has the same cost, denoted

$c(e)$. When this assumption fails we use cost functions, defined in the text. Other definitions for DCS— e.g., *minimum perfect DCS*, *minimum cost maximum cardinality DCS*, *minimum cost DCS*, follow by analogy with matching.

The *transportation problem* is to find a minimum cost perfect DCS in a bipartite multigraph where all edges have infinite multiplicity; alternatively if M is the maximum degree constraint, all multiplicities are M. If some multiplicities are less than M the problem is a *capacitated transportation problem*. (The usual definition of the transportation problem allows nonnegative real-valued degree constraints and edge multiplicities (both given multiplicities and those in the solution). This paper deals with the integral case of this problem. Note that if the given degree constraints and multiplicities are rational, they can be scaled up to integers. Also note that no loss of generality results from the constraint in this paper that the solution to the transportation problem has integral multiplicities— such an optimum solution always exists when the given degree constraints and multiplicities are integral [L]. Finally note that in our terminology the minimum perfect DCS problem is the same as the capacitated transportation problem).

Next consider a directed graph G with nonnegative edge costs. The *directed Chinese postman problem* is to find a multiplicity for each edge of G so the corresponding multigraph M is Eulerian, or if this is not possible M decomposes into the fewest possible number of open paths, and subject to this restriction, M has the smallest cost possible [PS]. This amounts to the following problem: Let $indegree(v)$ and $outdegree(v)$ be the number of edges directed *to* v and *from* v, respectively. Call vertex v a *start vertex* if $indegree(v) > outdegree(v)$, an *end vertex* if $indegree(v) < outdegree(v)$, and otherwise an *intermediate vertex*. The problem is to find a minimum cost, maximum cardinality set of paths P that start only at start vertices, end only at end vertices, each start vertex v starts at most $indegree(v) - outdegree(v)$ paths, and each end vertex v ends at most $outdegree(v) - indegree(v)$ paths. (The given graph G becomes an Eulerian multigraph if P contains p paths, for $p = \sum\{indegree(v) - outdegree(v) | v \text{ a start vertex}\} = \sum\{outdegree(v) - indegree(v) | v \text{ an end vertex}\}$).

2. Matching and extensions.

Section 2.1 presents our algorithm to find a minimum perfect matching on a bipartite graph. Section 2.2 gives extensions to other versions of matching, some facts about the algorithm needed in Section 3, and our shortest path algorithm. In this section n denotes the number of vertices in each vertex set V_0, V_1 of the given bipartite graph.

2.1. The matching algorithm.

For convenience assume the given graph G has a perfect matching (the algorithm can detect graphs not having a perfect matching, as indicated below).

The plan for the algorithm is to combine the Hungarian algorithm for weighted matching with the Hopcroft-Karp algorithm for cardinality matching. Recall that the Hungarian algorithm always chooses an augmenting path of smallest net cost. The Hopcroft-Karp algorithm always chooses an augmenting path of shortest length. Both of these rules can be approximated simultaneously *if* the costs are small integers. Arbitrary costs can be replaced by small integers by scaling. Thus our algorithm scales the costs. At each scale it computes a matching. The computation is efficient because it is similar to the Hopcroft-Karp algorithm; the matching is close to optimum because the computation is similar to the Hungarian algorithm. Now we give the details.

Each scale of the algorithm finds a close-to-minimum matching, defined as follows. Every vertex v has a *dual variable* $y(v)$. A *1-feasible matching* consists of a matching M and dual variables $y(v)$ such that for any edge vw,

$$y(v) + y(w) \le c(vw) + 1,$$
$$y(v) + y(w) = c(vw), \qquad \text{for } vw \in M.$$

A *1-optimal matching* is a perfect matching that is 1-feasible. If the +1 term is omitted from the first inequality, these are the usual complementary slackness conditions for a minimum perfect matching [L]. The following result is due to Bertsekas [Ber86].

Lemma 2.1. Let M be a 1-optimal matching.

(a) Any perfect matching P has $c(P) \ge c(M) - n$.

(b) If some integer k, $k > n$, divides each cost $c(e)$ then M is a minimum perfect matching.

Proof. Part (a) follows because $c(M) = \sum\{c(e) | e \in M\} = \sum\{y(v) | v \in V(G)\} \le c(P) + n$. Part ($b$) follows from ($a$) and the fact that any matching has cost a multiple of k. ∎

This lemma is the basis for the *main routine* of the algorithm, which does the scaling. The routine starts by computing a new cost $\overline{c}(e)$ for each edge e, equal to $n + 1$ times the given cost. Consider each $\overline{c}(e)$ to be a signed binary number $\pm b_1 b_2 \ldots b_k$ of $k = \lfloor \log(n+1)N \rfloor + 1$ bits. The routine maintains a variable $c(e)$ for each edge e, equal to its cost in the current scale. The routine initializes each $c(e)$ to 0 and each dual $y(v)$ to 0. Then it executes the following loop for index s going from 1 to k:

Step 1. For each edge e, $c(e) \leftarrow 2c(e)+$ (signed bit b_s of $\bar{c}(e)$). For each vertex v, $y(v) \leftarrow 2y(v)-1$.

Step 2. Call the *scale_match* routine to find a 1-optimal matching. ∎

Lemma 2.1(b) shows that the routine halts with a minimum perfect matching. Each iteration of the loop is called a *scale*. We give a *scale_match* routine that runs in $O(\sqrt{n}m)$ time. Since there are $O(\log(nN))$ scales this achieves the desired time bound.

It is most natural to work with small costs. The *scale_match* routine transforms costs to achieve this. Specifically, *scale_match* changes the cost of each edge vw to $c(vw)-y(v)-y(w)$; then it calls the *match* routine on these costs to find a 1-optimal matching M with duals $y'(v)$; then it adds $y'(v)$ to each dual $y(v)$ ($y(v)$ is the dual value before the call to *match*).

Clearly M with the new duals is a 1-optimal matching for cost function c. Further, since Step 1 of the main routine changes costs and duals so that the empty matching is 1-feasible, the costs input to *match* are integers -1 or larger. If vw is an edge in the 1-optimal matching found in the previous scale then after Step 1, $y(v)+y(w) \geq c(vw)-3$. Hence vw costs at most three in the costs for *match*. Thus there is a matching of cost at most $3n$. (This is true even in the first scale). We will show that if every edge costs at least -1 and a minimum perfect matching costs $O(n)$, *match* finds a 1-optimal matching in $O(\sqrt{n}m)$ time. This gives the desired time bound.

Note that the transformation done by *scale_match* is for conceptual convenience only. An actual implementation would not transform costs; rather *match* would work directly on the untransformed costs.

The *cost-length* of an edge e with respect to a matching M is

$$cl(e) = c(e) + \text{ (if } e \notin M \text{ then } 1 \text{ else } 0).$$

The *net cost-length* of a set of edges S with respect to M is

$$cl(S) = \sum\{cl(e) | e \in S - M\} \; - \sum\{cl(e) | e \in S \cap M\}.$$

This quantity equals the net cost of S (with respect to M) plus the number of unmatched edges in S. Hence an augmenting path with smallest net cost-length approximates both the smallest net cost augmenting path and the shortest augmenting path; this is in keeping with our plan for the algorithm.

An edge vw is *eligible* if $y(v)+y(w) = cl(vw)$, i.e., the 1-feasibility constraint for vw holds with equality. (Note that a matched edge is always eligible). It follows from the analysis below that an augmenting path of eligible edges has the smallest possible net cost-length. Hence the algorithm

augments along paths of eligible edges. If no such path exists it adjusts the duals to create one. The details are as follows.

Assume the costs given to *match* are integers at least -1, and there is a perfect matching costing at most an. (In the scaling algorithm $a = 3$).

procedure *match.*
Initialize all duals $y(v)$ to 0 and matching M to $\emptyset$. Then repeat the following steps until Step 1 halts with the desired matching:

Step 1. Find a maximal set $\mathcal{A}$ of vertex disjoint augmenting paths of eligible edges. For each path $P \in \mathcal{A}$, augment the matching along P, and for each vertex $w \in V_1 \cap P$, decrease $y(w)$ by 1. (This makes the new matching M 1-feasible). If the new matching M is perfect, halt.

Step 2. Do a Hungarian search to adjust the duals (maintaining 1-feasibility) and find an augmenting path of eligible edges. ∎

We now give the details of Steps 1 and 2 that are needed to analyze *match*. (A full description of these steps is given later). Both steps can be implemented in $O(m)$ time. Step 2 is a Hungarian search (essentially Dijkstra's shortest path algorithm, see e.g. [L]). The search does a number of *dual adjustments*. Each dual adjustment calculates a positive integer δ and increases or decreases various dual values by δ, so as to preserve 1-feasibility and eventually create an augmenting path of eligible edges. (The dual adjustment is defined more precisely below). At any point in the algorithm define

F = the set of free vertices in V_0;

Δ = the sum of all dual adjustment quantities δ
in all Hungarian searches so far.

The Hungarian search maintains the duals so that any free vertex $v \in F$ has $y(v) = \Delta$ and any free vertex $v \in V_1$ has $y(v) = 0$.

To analyze the *match* routine, first observe that it is correct: The changes to the matching (in Step 1) and to the duals (in Step 2) keep M a 1-feasible matching. If M is not perfect but G has a perfect matching, the Hungarian search creates an augmenting path of eligible edges. Hence the algorithm eventually halts with M a 1-optimal matching, as desired. (Note that if G does not have a perfect matching, this is eventually detected in Step 2).

To analyze the run time, consider any point in the execution of *match*. Let M be the current matching, and define F and Δ as above. Let M^* be a minimum perfect matching. For any set of

To analyze the run time, consider any point in the execution of *match*. Let M be the current matching, and define F and Δ as above. Let M^* be a minimum perfect matching. For any set of edges S let $cl(S)$ denote net cost-length with respect to M.

$M^* \oplus M$ consists of an augmenting path P_v for each $v \in F$, plus alternating cycles C_w. Thus

$$n + c(M^*) - c(M) \geq cl(M^* \oplus M) = \sum_{v \in F} cl(P_v) + \sum_w cl(C_w). \tag{1}$$

To estimate the right-hand side, consider an alternating path P from $u \in V_0$ to $m \in V_0$, where u is on an unmatched edge of P and m is on a matched edge of P (m stands for "matched"; no confusion should result from the double usage of m). Then

$$y(u) \leq y(m) + cl(P). \tag{2}$$

This follows since for edges $uv \notin M$ and $vm \in M$, $y(u)+y(v) \leq cl(uv)$ and $y(v)+y(m) = cl(vm)$, so $y(u) \leq y(m) + cl(uv) - cl(vm)$. Inequality (2) implies that any alternating cycle C has $cl(C) \geq 0$. It also implies that any augmenting path P_v from some $v \in F$ to some free vertex $t \in V_1$ has $y(v) + y(t) \leq cl(P_v)$. Recall that the Hungarian search keeps $y(v) = \Delta$ and $y(t) = 0$. Hence $\Delta \leq cl(P_v)$, and the right-hand side of (1) is at least $|F|\Delta$.

By assumption on the input to *match*, $c(M^*) \leq an$ and $c(M) \geq -n$. Hence the left-hand side of (1) is at most bn for $b = a + 2$. Thus we have shown

$$|F|\Delta \leq bn. \tag{3}$$

This implies there are $O(\sqrt{n})$ iterations of the loop of *match*. To see this note that each execution of Step 1 (except possibly the first) augments along at least one path, because of the preceding Hungarian search. Hence at most $\sqrt{bn} + 1$ iterations start with $|F| \leq \sqrt{bn}$. From (3), $|F| \geq \sqrt{bn}$ implies $\Delta \leq \sqrt{bn}$. We will show that each Hungarian search increases Δ by at least one. This implies at most $\sqrt{bn} + 1$ iterations start with $\Delta \leq \sqrt{bn}$, giving the desired bound.

Now we show that a Hungarian search S increases Δ by at least one. It suffices to show that S does a dual adjustment (since any dual adjustment quantity δ is a positive integer). Search S does a dual adjustment unless when it starts, there is an augmenting path P of eligible edges. Clearly P intersects some augmenting path found in Step 1. It is easy to see that P contains an unmatched edge vw, with w but not v in an augmenting path of Step 1, and $w \in V_1$. But when Step 1 decreases $y(w)$ this makes vw ineligible. So P does not exist, and S does a dual adjustment.

In summary *match* does $O(\sqrt{n})$ iterations. Each iteration takes time $O(m)$, giving the desired time bound $O(\sqrt{n}m)$.

It remains to give the details of Steps 1 and 2. Step 1 finds the augmenting paths P by depth-first search. To do this it marks every vertex reached in the search. It initializes a path P to a free unmarked vertex of V_0. To grow P it scans an eligible edge xy from the last vertex x of P (x will always be in V_0). If y is marked, the next eligible edge from x is scanned; if none exists the last two edges of P (one matched and one unmatched) are deleted from P; if P has no edges another path is initialized. If y is free another augmenting path has been found; y is marked, the path is added to $\mathcal{A}$ and the next path is initialized. The remaining possibility is that y is matched to a vertex z. In this case y and z are marked, edges xy, yz are added to P and the search is continued from z.

It is clear that this search uses $O(m)$ time. To show that it halts with $\mathcal{A}$ maximal, first observe that for any marked vertex $x \in V_0 - V(\mathcal{A})$, every eligible edge xy has y marked and matched, or in $V(\mathcal{A})$. (Note that $V(\mathcal{A})$ is the set of vertices in paths of $\mathcal{A}$). Hence an easy induction shows that an alternating path of eligible edges, starting at a free vertex of V_0 and vertex disjoint from $\mathcal{A}$, has all its V_0 vertices marked, and is not augmenting.

Step 2 is the Hungarian search. It grows a forest $\mathcal{F}$ of eligible edges, from roots F. A pair of eligible edges vw, ww', where $v \in V_0 \cap \mathcal{F}$, $w \notin \mathcal{F}$, and $ww' \in M$, is added to $\mathcal{F}$ whenever possible. Eventually either an augmenting path of eligible edges is found or $\mathcal{F}$ cannot be enlarged.

In the latter case a *dual adjustment* is done. It changes duals in a way that preserves 1-feasibility and allows $\mathcal{F}$ to be enlarged, as follows. It computes the dual adjustment quantity

$$\delta = \min\{cl(vw) - y(v) - y(w) | v \in V_0 \cap \mathcal{F},\ w \notin \mathcal{F}\}.$$

For each $v \in \mathcal{F}$, it increases $y(v)$ by $\delta \times$ (if $v \in V_0$ then 1 else -1). It is easy to see that this achieves the goal of the dual adjustment (any edge vw achieving the above minimum becomes eligible after the dual adjustment, and so can be added to $\mathcal{F}$).

After the dual adjustment the search continues by enlarging $\mathcal{F}$. Eventually the desired augmenting path is found.

Note that as claimed above, at any point in the algorithm a free vertex v has $y(v) = \Delta$ if $v \in F$ (since every dual adjustment increases $y(v)$) and $y(v) = 0$ if $v \in V_1$ (no dual adjustment changes $y(v)$).

A Hungarian search can be implemented in $O(m)$ time. This depends on two observations. First, the proper data structure allows a dual adjustment to change all duals $y(v)$ in $O(1)$ time total. Specifically the algorithm keeps track of Δ (defined above). When a vertex v is added to $\mathcal{F}$ its current dual value and the current value of Δ are saved as $y^0(v)$ and $\Delta(v)$, respectively. Then

at any time the current value of $y(v)$ can be calculated as

$$y^0(v) + (\Delta - \Delta(v)) \times (\text{if } v \in V_0 \text{ then 1 else } -1).$$

Hence the dual adjustment is accomplished by simply increasing the value of Δ.

The second observation is how to compute δ in a dual adjustment. The usual implementation of a Hungarian search does this with a priority queue that introduces a logarithmic factor into the time bound [e.g., FT]. This can be avoided when, as in our case, costs are small integers (this was observed in [D, W] for Dijkstra's shortest path algorithm). The details are as follows. The next value of δ is the amount that the next value of Δ increases from its current value. Hence it suffices to calculate the next value of Δ. The next value of Δ is the smallest possible value such that some edge vw with $v \in V_0 \cap \mathcal{T}$ and $w \notin \mathcal{T}$ becomes eligible (when duals are adjusted by δ). Thus the next value of Δ equals

$$\min\{cl(vw) - y^0(v) - y(w) + \Delta(v) | v \in V_0 \cap \mathcal{T},\ w \notin \mathcal{T}\}.$$

Since any Hungarian search has $|F| \geq 1$, inequality (3) implies $\Delta \leq bn$. The algorithm maintains an array $Q[1..bn]$. Each entry $Q[r]$ points to a list of edges vw that can make $\Delta = r$, i.e., $v \in V_0 \cap \mathcal{T}$, $w \notin \mathcal{T}$ and $r = cl(vw) - y^0(v) - y(w) + \Delta(v)$. The algorithm scans down Q and chooses next value of Δ as the smallest value r with $Q[r]$ nonempty. This gives the next value of δ, and the newly eligible edges, as desired. The total overhead for scanning is $O(n)$ since Q has bn entries. (Note that an edge vw with $v \in V_0 \cap \mathcal{T}$, $w \notin \mathcal{T}$ does not get entered in this data structure if $cl(vw) - y^0(v) - y(w) + \Delta(v) > bn$).

Only one detail of the derivation remains: We have assumed that the dual values $y(v)$ do not grow too large, so that arithmetic operations use $O(1)$ time. To justify this we show that each $y(v)$ has magnitude $O(n^2N)$. It suffices to do this for $v \in V_0$. Define Y_s as $\max\{|y(v)||v \in V_0\}$ after the s^{th} scale. Then $Y_0 = 0$ and $Y_{s+1} \leq 2Y_s + bn - 1$ (since $\Delta \leq bn$). Thus $Y_k \leq (2^k - 1)(bn - 1) = O(n^2N)$ as desired. Note that the input uses a word size of at least $\max\{\log N, \log n\}$ bits. Hence at worst the algorithm uses triple-word integers for the dual variables.

Theorem 2.1. A minimum perfect matching on a bipartite graph can be found in $O(\sqrt{n}m\log(nN))$ time and $O(m)$ space. ∎

A heuristic that may speed up the algorithm in practice is to prune the graph at the start of each scale. Specifically, *scale_match* can delete any edge whose new cost is $6n$ or more. In proof,

recall that in the costs computed by *scale_match* there is a perfect matching M costing at most $3n$; taking into account the low order bits of cost that are not included in the current scale, the true cost of M is less than $4n$. In the costs computed by *scale_match* every edge costs at least -1; again taking into account low order bits, the true cost is more than -2. Hence a matching containing an edge of new cost $6n$ or more has true cost more than $4n$ and so is not minimum.

2.2. Extensions of the matching algorithm.

The bounds of Theorem 2.1 also apply to finding a minimum cost matching. To see this let G be the given graph. Form $\overline{G}$ by taking two copies of G; for each $v \in V(G)$ join the two copies of v by a cost zero edge. Then $\overline{G}$ is bipartite, and a minimum perfect matching on $\overline{G}$ gives a minimum cost matching on G.

A similar result holds for minimum cost maximum cardinality matching. The construction is the same except that the edges joining two copies of v cost nN. The problem of finding a minimum cost matching of given cardinality can also be solved in the same bounds; it is most convenient to use Theorem 3.2 below.

Returning to perfect matching, several properties of *match* are needed for Section 3. Define

$$A = \text{ the total length of all augmenting paths found by } match.$$

We first derive a bound on A. Let P_i denote the i^{th} augmenting path found by *match*. Let ℓ_i be its length, measured as its number of unmatched edges; let Δ_i denote the value of Δ when P_i is found; let M_i be the matching after augmenting along P_i. Recall that in the Hopcroft-Karp algorithm, for some constant c, $\ell_i \leq cn/(n-i+1)$. Thus the total augmenting path length is $\sum_{i=1}^{n} \ell_i = O(n \log n)$ [ET]. In *match*, ℓ_i does not have a similar bound. However it is bounded in an amortized sense, as follows.

Lemma 2.2. For any k in $[1..n]$, $\sum_{i=1}^{k} \ell_i + c(M_k) = \sum_{i=1}^{k} \Delta_i$.

Proof. A calculation similar to (2) shows that for any i, $\Delta_i = cl(P_i)$. It is easy to see that $cl(P_i) = \ell_i + c(M_i) - c(M_{i-1})$ (assume $c(M_0) = 0$). Summing these relations gives the lemma. ∎

Corollary 2.1. $A = O(n \log n)$.

Proof. Since $|M_k| = k$ the entry conditions for *match* imply $c(M_k) \geq -k$. Hence $A \leq n + \sum_{i=1}^{n} \Delta_i$. By (3), $\Delta_i \leq bn/(n-i+1)$. Summing these inequalities gives the lemma. ∎

The second property shows that the depth-first search of Step 1 never encounters a cycle. A similar property for network flows is used in [GoT87a].

Lemma 2.3. In *match* there is never an alternating cycle of eligible edges.

Proof. Initially there are no matched edges, so there are no alternating cycles of eligible edges. In a Hungarian search, whenever the duals of a matched edge vw are changed, $w \in V_1$ gets $y(w)$ decreased. Hence any edge joining w to a vertex not in the search forest $\mathcal{F}$ is ineligible. This implies the Hungarian search does not create an alternating cycle of eligible edges. Similar reasoning applies when an augment creates a new matched edge and changes duals. ∎

Some applications of matching require ordinary dual variables, defined as follows. The duals are *dominated* on edge vw if $y(v) + y(w) \leq c(vw)$; they are *tight* on vw if equality holds. The duals are *dominated and tight* for a given matching if each edge is dominated and each matched edge is tight; such duals are the usual linear programming dual variables [L]. The scaling algorithm halts with 1-optimal duals, but these are not necessarily dominated and tight. Such duals can be found as follows.

Let G^+ be G with an additional vertex $s \in V_0$ and an edge sv for each $v \in V_1$. Extend the given cost function c to G^+ by defining $c(sv)$ as an arbitrary integer; the cost function used by the matching algorithm extends to G^+ by its definition, $\bar{c} = (n+1)c$. To specify a cost function on G^+ we write $G^+; c$ or $G^+; \bar{c}$. Let M be a minimum perfect matching on G; for vertex v let v' denote its mate, i.e., $vv' \in M$. For $v \in V_0$ let M_v be a minimum perfect matching on $G^+ - v; c$. (Such a matching exists, for instance $M - vv' + sv'$). Set

$$y(v) = \textbf{if } v \in V_0 \textbf{ then } -c(M_v) \textbf{ else } c(vv') - y(v').$$

These duals are dominated and tight on G. (This can be proved by an argument similar to the algorithm given below. Alternatively see [G87] for a proof from first principles).

Suppose a Hungarian search (as in *match*) is done on $G^+; \bar{c}$. It halts with a tree T of eligible edges, rooted at s. Clearly T is a spanning tree. For any $v \in V_0$, augmenting along the sv-path in T gives a 1-optimal matching N_v on $G^+ - v; \bar{c}$. N_v is a minimum perfect matching on $G^+ - v; c$. This follows from Lemma 2.1, since $G^+ - v$ and G have the same number of vertices. Hence N_v qualifies as M_v.

In summary, the following procedure finds dominated tight duals. Given is the output of the matching algorithm, i.e., a 1-optimal matching on $G;\bar{c}$ with duals y. Form $G^+;\bar{c}$, defining $c(sv) = \lceil y(v)/(n+1) \rceil$ for each $v \in V_1$; also set $y(s) \leftarrow 0$ (this gives 1-feasible duals). Do a Hungarian search to construct a spanning tree T of eligible edges rooted at s. Do a depth-first search of T to find $c(M_v)$ for each $v \in V_0$. Define dominated tight duals $y(v)$ by the above formula.

The time for this algorithm is $O(m)$. This is clear except perhaps for the time for the Hungarian search. The choice of $c(sv)$ ensures that $\Delta \leq n$. Hence, as in *match*, the Hungarian search can be implemented using an array Q. This gives $O(m)$ time.

Corollary 2.2. Dominated and tight duals on a bipartite graph can be found in $O(\sqrt{n}m\log(nN))$ time and $O(m)$ space. ∎

This implies the next result. Consider a directed graph with n vertices, m edges, and arbitrary (possibly negative) edge lengths.

Theorem 2.2. The single-source shortest path problem on a directed graph with arbitrary edge lengths can be solved in $O(\sqrt{n}m\log(nN))$ time and $O(m)$ space.

Proof. This problem can be solved by finding dominated tight duals on a bipartite graph whose costs are the edge lengths, and then running Dijkstra's algorithm [G85a]. ∎

Obviously the same bound holds for $O(\sqrt{n})$ sources.

3. Degree-constrained subgraphs and extensions.

This section extends the matching algorithm to derive the last four bounds of Table I. Section 3.1 gives an algorithm for the minimum perfect degree-constrained subgraph problem, deriving time bounds for finding a degree-constrained subgraph and for solving the transportation problem. Section 3.2 discusses scaling edge multiplicites, which improves the bounds when edge multiplicities are large. Section 3.3 extends the results to network flow. Throughout Section 3, n denotes the number of vertices in the input graph.

3.1. The degree-constrained subgraph algorithm.

This subsection gives an algorithm for the perfect degree-constrained subgraph problem. Recall that a multigraph has m edges and $\overline{m}$ edges counting multiplicities. Note that a perfect DCS problem on a multigraph of n vertices and $\overline{m}$ edges can be reduced in linear time to a perfect matching problem on a graph of $O(\overline{m})$ vertices and edges [G87]. Hence Theorem 2.1 immediately implies a bound of $O(\overline{m}^{3/2}\log(\overline{m}N))$ for the DCS problem. We now derive the better bound given in Table I.

For a DCS D, the *cost-length* of edge e is

$$cl(e) = c(e) + (\textbf{if } e \notin D \textbf{ then } 1 \textbf{ else } 0).$$

A *1-feasible DCS* is a DCS D and dual variables $y(v)$ for each vertex v, such that for any edge vw,

$$y(v) + y(w) \leq cl(vw), \qquad \text{for } vw \notin D,$$
$$y(v) + y(w) \geq cl(vw), \qquad \text{for } vw \in D.$$

A *1-optimal DCS* is a perfect DCS that is 1-feasible. (Note that the definition of a 1-feasible matching is slightly different— the second relation holds with equality. The difference is not significant: if we treat a matching problem as a DCS problem, a 1-feasible DCS gives a 1-feasible matching, by lowering duals as necessary to achieve the desired equalities).

As in Lemma 2.1, if every cost is divisible by k, $k > n/2$, then a 1-optimal DCS is a minimum perfect DCS. This is essentially a result of Bertsekas [Ber86]. In proof, note that a perfect DCS D has minimum cost if any alternating (simple) cycle C has $c(C \cap D) \leq c(C - D)$. This condition can be verified for a 1-optimal DCS D by a calculation similar to Lemma 2.1.

Now we describe the algorithm. Many details are exactly as in Section 2, so we elaborate only on the parts that change. All data structures have size $O(m)$. Clearly the multigraph G can be represented by such a structure.

The main routine works in (at most) $\lfloor \log(n+2)N \rfloor$ scales. (This is justified by the above analog of Lemma 2.1; each original cost is multiplied by $\lceil (n+1)/2 \rceil$). Steps 1-2 and *scale_match* are unchanged. Let D_0 be the 1-optimal DCS of the previous scale. Note that the *match* routine is called with integral costs $c(e)$ that are at least -1 for $e \notin D_0$ and at most three for $e \in D_0$.

The *match* routine initializes all duals $y(v)$ to 0 and D to $\{e | c(e) < -1\}$. (Clearly D does not violate any degree constraint). The definition of an *eligible* edge vw is still $y(v) + y(w) = cl(vw)$. Step 1 of *match* finds a maximal set of augmenting paths of eligible edges and augments the matching along each path. (Unlike Section 2, no duals are changed after an augment; the new DCS is 1-feasible and the edges on an augmenting path become ineligible). Step 2 does a Hungarian search

to adjust duals and find an augmenting path of eligible edges. Note this algorithm is correct: Since the Hungarian search maintains 1-feasiblity, the algorithm halts with a 1-optimal DCS (assuming a perfect DCS exists).

Step 1 is implemented by a depth-first search similar to Section 2, modified for degree constraints larger than one: Each augmenting path P is initialized to a vertex $x \in V_0$ with positive deficiency; x is used to initialize paths P until its deficiency becomes zero or it is deleted from P. P is grown as an alternating path, so that when its last vertex x is in V_0 an edge not in D is scanned, and when x is in V_1 an edge of D is scanned. Instead of vertex marks, each vertex has a pointer to its last unscanned edge. The last edge of P gets deleted if x has no more unscanned edges. It is easy to see the time for Step 1 is $O(\overline{m})$. (As shown below each augmenting path is simple, although this fact is not needed for correctness).

The details of the Hungarian search are similar to Section 2. The main differences stem from the fact that the search forest $\mathcal{F}$ is grown edge-by-edge, rather than in pairs of unmatched and matched edges. The time for the search is $O(m)$. This assumes that, as in Section 2, an array $Q[1..dn]$ is used to compute minima; here d is the constant of Lemma 3.3, which justifies using the array.

This completes the description of the DCS algorithm. The discussion shows that it is correct. The efficiency analysis uses three inequalities, each analogous to (3) of Section 2. We use notation similar to Section 2: D is the DCS at any point in *match*. D_0 is the 1-optimal DCS of the previous scale; hence each of its edges costs at most $a = 3$. F is the set of vertices in V_0 with positive deficiency; Φ is their total deficiency,

$$\Phi = \sum\{\phi(v, D) | v \in F\}.$$

Δ is the sum of all dual adjustment quantities δ in all Hungarian searches so far. Each $x \in F$ has $y(x) = \Delta$. P_x denotes any one of the augmenting paths containing x in $D_0 \oplus D$.

Lemma 3.1. For some constant b, at any point in *match*, $\Phi\Delta \leq bU$.

Proof. The argument of Section 2 gives an analog of (1),

$$cl(D_0 \oplus D) \geq \sum\{\phi(v, D)y(v) | v \in F\}.$$

An edge of $D_0 - D$ has cost-length at most $a + 1$; an edge of $D - D_0$ has cost-length at least -1. Hence the lemma holds with $b = (a + 2)/2$. ∎

The second inequality is for graphs with bounded multiplicity. It generalizes [ET]. Recall that M denotes the maximum multiplicity of an edge in the multigraph.

Lemma 3.2. For some constant c, at any point in *match*, $\Delta\sqrt{\Phi} \le cn\sqrt{M}$.

Proof. Set $b = a + 2$. For each integer j define

$$U_j = \{u \in V_0 | y(u) \in [b(j-1)..bj)\},$$
$$W_j = \{w \in V_1 | y(w) - z - 1 \in (-bj.. - b(j-1)]\}.$$

We will show that for any $j \in [1..\lceil \Delta/b + 1 \rceil]$, each P_z has an edge uw with $u \in U_j$, $w \in W_j$. This implies $M|U_j||W_j| \ge \Phi$. Thus $|U_j|$ or $|W_j|$ is at least $\sqrt{\Phi/M}$. Hence $n \ge \sqrt{\Phi/M}(\Delta/b)$, as desired.

To find the desired edge uw in P_z, let the edges in $P_z - D$ be u_iw_i, $i = 1, \ldots, k$ (thus $u_1 = z$, and $u_{i+1}w_{i+1}$ follows u_iw_i). Since $P_z \subseteq D_0 \oplus D$,

$$\begin{aligned} y(u_i) + y(w_i) &\le a + 1, \\ y(w_i) + y(u_{i+1}) &\ge -1. \end{aligned} \tag{4}$$

Note that $y(u_1) = \Delta$; $y(u_k) < b$ (by (4) and $y(w_k) = 0$); and $y(u_{i+1}) \ge y(u_i) - b$ (also by (4)). These three inequalities imply that for any $j \in [1..\lceil \Delta/b + 1 \rceil]$, P_z has some $u_i \in U_j$. For a given j choose the last such i. Then $u_{i+1} \in U_{j-1}$. Together with (4) this implies $w_i \in W_j$, since

$$-bj < -y(u_{i+1}) - b \le y(w_i) - a - 1 \le -y(u_i) \le -b(j-1).$$

We have shown that $w_i \in W_j$ and $u_i \in U_j$, as desired. ∎

Before continuing we give a useful refinement of Lemma 3.2. Let X be a matching such that every edge not in X has multiplicity at most M_X.

Corollary 3.1. For some constant c, at any point in *match*, $\Delta\sqrt{\Phi} \le cn\sqrt{M_X}$.

Proof. The proof is similar to the lemma. We show that for any $j \in [1..\lceil \Delta/b + \bar{1} \rceil]$, each P_z has an edge uw not in X with $u \in U_j \cup U_{j-1}$, $w \in W_j$. This implies $M_X|U_j \cup U_{j-1}||W_j| \ge \Phi$, which leads to the desired conclusion.

To find the desired edge uw for a path P_z, proceed exactly as in the lemma to find an index i with $u_i \in U_j$, $u_{i+1} \in U_{j-1}$ and $w_i \in W_j$. One of the edges u_iw_i, w_iu_{i+1} is not in X and can be taken as uw. ∎

Another bound on Δ is useful for large multiplicities. It is similar to the bound used in [GoT87a]. It justifies using the array $Q[1..dn]$ to compute minima in the Hungarian search.

Lemma 3.3. For some constant d, at any point in *match*, $\Delta \leq dn$.

Proof. The proof of Lemma 3.2 shows that for any $j \in [1..\lceil \Delta/b \rceil + 1]]$, P_z has some $u \in U_j$. ∎

Corollary 3.2. The number of iterations of *match* is $O(\min\{\sqrt{U}, n^{2/3}M^{1/3}, n\})$.

Proof. Each execution of Step 1 (except possibly the first) augments along at least one path, i.e., it decreases Φ by at least one. The definition of Step 1 implies that each Hungarian search (except the last) increases Δ by at least one. Now the first two bounds of the lemma follow because at any point in the algorithm Δ or Φ is at most B, where Lemma 3.1 gives $B = \sqrt{bU}$ and Lemma 3.2 gives $B = (cn)^{2/3}M^{1/3}$. The third bound follows from Lemma 3.3. ∎

The corollary implies the following time estimates. The estimates are good for graphs or multigraphs of very small multiplicity.

Theorem 3.1. A minimum perfect DCS on a bipartite multigraph can be found in $O(\min\{\sqrt{U}, n^{2/3}M^{1/3}\}m \log(nN))$ time. The space is $O(m)$. ∎

For example in a bipartite graph a minimum perfect DCS can be found in $O(\min\{\sqrt{m}, n^{2/3}\}m \log(nN))$ time.

The bounds of the theorem also apply to finding a minimum cost DCS. To see this let G be the given multigraph or graph. Form $\overline{G}$ by taking two copies of G and adding a set of edges X, where for each $v \in V(G)$, X contains an edge joining the two copies of v, with multiplicity $u(v) - \ell(v)$ and cost zero. It is easy to see that $\overline{G}$ is bipartite, and a minimum perfect DCS on $\overline{G}$ gives a minimum cost DCS on G. Furthermore X is a matching, so Corollary 3.1 applies with $M_X = M$. This implies the time bound of the theorem for minimum cost DCS. Minimum cost maximum cardinality DCS is similar.

Now we derive bounds that are good for multigraphs with moderately sized multiplicities. First observe Lemma 2.3 still holds: in *match* there is no alternating cycle of eligible edges. The proof is essentially the same: There is no such cycle initially, since the edges in D are ineligible. A Hungarian search does not create such a cycle, since immediately after a dual adjustment a cycle leaving $\mathcal{T}$ on a new eligible edge re-enters $\mathcal{T}$ on an ineligible edge.

This fact ensures that the time for a depth-first search in Step 1 is $O(m)$ plus the total augmenting path length. Thus the total time for *match* is $O(mB + A)$, where B is the number of iterations and A is the total augmenting path length. Corollary 3.2 bounds B; now we estimate A.

Lemma 3.4. $A = O(\min\{U \log U, n\sqrt{MU}\})$.

Proof. As in Corollary 2.1, $A \leq U + \sum_{i=1}^{U} \Delta_i$. For the first bound estimate the summation as in Corollary 2.1, using Lemma 3.1. For the second bound, Lemma 3.2 shows that the summation is at most $\sum_{i=1}^{U} cn\sqrt{M/i} = O(n\sqrt{MU})$. ∎

Theorem 3.2. The transportation problem (capacitated or not) can be solved in $O((\min\{\sqrt{U}, n^{2/3}M^{1/3}, n\}m + \min\{U \log U, n\sqrt{MU}\}) \log(nN))$ time. The space is $O(m)$. ∎

To understand this rather involved time bound, first note that the terms containing M are relevant only in the capacitated transportation problem. The main use of the theorem in this paper is when $U = O(nm)$, in which case the time is $O(nm \log n \log(nN))$; this bound is used in Section 3.2 to solve transportation problems with larger U. For further applications we concentrate on the range $M = O(n)$. In this case the above bound for $U = O(nm)$ holds, and also the bound $O(n^2\sqrt{m} \log(nN))$; hence in this range the performance is competitive with [GoT87a]. In most of the range $M = O(n)$ the bounds of Theorem 3.2 are those of Theorem 3.1 with m replaced by m: Using $U \log U$ as the second term of the time bound and writing Bm as the first term, the first term dominates if $U = O(Bm/\log n)$. Hence the bound is $O(n^{2/3}M^{1/3}m \log(nN))$ if $U = O(n^{2/3}M^{1/3}m/\log n)$, e.g., $M = O(n/(\log n)^{3/2})$; the bound is $O(\sqrt{mM}m \log(nN))$ if $U = O(\sqrt{mM}m/\log n)$, e.g., $M = O(m/(\log n)^2)$; the bound is $O(\sqrt{nM}m \log(nN))$ if $U = O(\sqrt{nM}m/\log n)$, e.g., all degree constraints are $O(M)$ and $M = O(m^2/(n(\log n)^2)$.

As in Theorem 3.1, the same bounds hold for networks where each node has an upper and lower bound on its desired degree, and the objective is minimum cost or minimum cost maximum cardinality.

3.2. Scaling edge multiplicities.

In multigraphs with large multiplicities efficiency is gained by scaling the multiplicities. Let D be a DCS. Recall that for an edge e, $u(e)$ and $d(e)$ denote the multiplicities of e in G and D, respectively; for a vertex v, $u(v)$ and $d(v)$ denote the degree constraint of v and the degree of v

in D, respectively. The term *u-value* refers to a multiplicity $u(e)$ or a degree constraint $u(v)$. The approach is to scale u-values. The "closeness lemma" needed for scaling is the following.

Let G be a multigraph with u-values for which D is a minimum cost maximum cardinality DCS. Form u^+ by adding one to the u-values of an arbitrary subset of vertices and edges (in particular a u-value can increase from zero to one). Let I be the number of increased u-values (so $I \leq m+n$). Let D^+ be a minimum cost maximum cardinality DCS for u^+. Let $D^+ \oplus D$ denote the subgraph that is the direct sum of subgraphs D^+ and D (i.e., for any edge e, $D^+ \oplus D$ has $|d^+(e) - d(e)|$ copies of e). Choose D^+ so that $|D^+ \oplus D|$ is as small as possible. $\phi^+(v, D)$ denotes the deficiency of D at v for u-values u^+.

Lemma 3.5. *$D^+ \oplus D$ can be partitioned into at most I simple alternating paths and cycles (where "alternating" means edges are alternately in D and D^+).*

Proof. Since both D^+ and D are DCS's for u^+, $D^+ \oplus D$ can be partitioned into simple alternating paths and cycles; for each vertex v, at most $\phi^+(v, D^+)$ paths end at v on a D-edge, and similarly for a D^+-edge. Call an edge vw with $d^+(vw) > d(vw)$ *new* if either

(i) $d(vw) = u(vw)$, or

(ii) vw is an end of a path of $D^+ \oplus D$ and $d(v) = u(v)$.

There are at most I new edges. (A type (i) new edge clearly has an increased u-value. For a type (ii) new edge vw, v has an increased u-value and $\phi^+(v, D) = 1$, so vw is the only type (ii) edge associated with v). Thus it suffices to show that any alternating path or cycle P of $D^+ \oplus D$ contains a new edge.

P does not begin and end with a D-edge, since D^+ has maximum cardinality. Suppose P does not contain a new edge. Then $D \oplus P$ is a feasible DCS for u. (This follows since a D^+-edge vw of P has $d(vw) < u(vw)$; further if this edge vw is an end of P then $d(v) < u(v)$). P does not begin and end with a D^+-edge, since D has maximum cardinality. Thus P is an even length alternating path or cycle. This implies P has zero net cost (with respect to D or D^+). But this contradicts the fact that $|D^+ \oplus D|$ is as small as possible. ∎

The lemma indicates that $D^+ \oplus D$ can be found in a "small" multigraph G', defined as follows. A vertex $v \in V(G)$ corresponds to v_1, $v_2 \in V(G')$; G' has an edge v_1v_2 of cost 0 and multiplicity I. An edge $vw \in E(G)$ corresponds to edges v_1w_1, $v_2w_2 \in E(G')$, with multiplicities and costs

$$u'(v_1w_1) = u^+(vw) - d(vw), \quad c'(v_1w_1) = c(vw) - nN;$$
$$u'(v_2w_2) = d(vw), \quad c'(v_2w_2) = -c(vw) + nN.$$

Call these edges *G-edges*; edges v_2w_2 are *D-edges* and edges v_1w_1 are *non-D-edges*. Finally each $v \in V(G)$ has upper and lower degree constraints $u'(v_1) = u'(v_2) = I$, $\ell'(v_1) = 0$, $\ell'(v_2) = I - \phi^+(v, D)$.

Consider D', a minimum cost DCS of G'. It is easy to see that the G-edges of D' can be partitioned into at most I paths and cycles that are alternating for D, and that $D \oplus D'$ is a feasible DCS (note that the lower bounds in G' allow a vertex v to be on at most $\phi^+(v, D)$ non-D-edges at the end of an alternating path). Furthermore, the costs of G-edges guarantee that D' has as many augmenting paths as possible and no "de-augmenting paths" (i.e., paths that begin and end with D-edges). Thus D' is the desired set of paths ($D^+ \oplus D$ or an equivalent set).

Now we state the *capacity scaling algorithm* for finding a minimum perfect DCS. Given a DCS problem on a multigraph G, let $\bar{u}$ denote the given u-values, with M the largest $\bar{u}$-value. (Without loss of generality M is the $\bar{u}$-value of a vertex). Consider each $\bar{u}$-value to be a binary number $b_1 \ldots b_k$ of $k = \lfloor \log M \rfloor + 1$ bits. The routine maintains u as the u-values in the current scale, and D (and d) as the DCS in the current scale. It initializes each $u(e)$, $d(e)$ and $u(v)$ to zero. Then it executes the following loop for scale index s going from 1 to k:

Step 1. For each edge e, $d(e) \leftarrow 2d(e)$ and $u(e) \leftarrow 2u(e)+$ (bit b_s of $\bar{u}(e)$). For each vertex v, $u(v) \leftarrow 2u(v)+$ (bit b_s of $\bar{u}(v)$).

Step 2. Form the multigraph G' described above. (Note $I \leq m + n$; u^+ is given by u in the algorithm.)

Step 3. Let D' be a minimum cost DCS on G'. Set $D \leftarrow D \oplus D'$ and let d be the function corresponding to D. ∎

The correctness of this algorithm follows from the above discussion. (Note that this algorithm works on both bipartite and general graphs). To analyze the running time, assume that the DCS in Step 3 is found using the algorithm of Theorem 3.2 (note that $U = O(nm)$). The following bound is immediate.

Theorem 3.3. The transportation problem (capacitated or not) can be solved in $O(nm \log n \log(nN) \log M)$ time. The space is $O(m)$. ∎

This result extends to the variants of the perfect DCS problem mentioned above.

The capacity scaling algorithm has a variant, called *EK capacity scaling*, since it is similar to the mincost flow algorithm of Edmonds and Karp [EK]. EK capacity scaling is used below to solve flow problems with lower bounds. It maintains a set of dominated tight duals on G for u and

D. ("Dominated tight" duals satisfy the inequalities for 1-feasiblity with cost-length cl replaced by cost c). The algorithm scales up costs and duals exactly as in Step 1 of the main routine and *scale_match*. This ensures that the costs input to *match* are positive integers except for edges that were in the DCS of the previous scale.

The *match* routine initializes all duals $y(v)$ to 0 and the DCS to $\{e|c(e) \leq 0\}$. This DCS does not violate any degree constraint and the duals are dominated tight. Then *match* does as many minimum cost augmentations as possible. The augmenting paths are found using the Hungarian algorithm. Then *match* searches for alternating even paths and cycles with minimum negative cost, and augments along them ("augmenting along a path or cycle" is defined by analogy with augmenting along an augmenting path; such an augmentation, in the current context, gives a new matching of equal cardinality). These searches are also implemented with the Hungarian algorithm. (A minor point is that *match* uses the standard Hungarian algorithm, based on dominated tight duals as in [L]. This differs from the Hungarian search in Section 3.1 which uses 1-feasible duals. The main difference is the definition of "eligible", which uses either cost or cost-length as appropriate).

Each Hungarian search takes time $O(m + n \log n)$ using Fibonacci heaps [FT]. In problems where each scale has $I = O(n)$, the total time for EK capacity scaling is $O(n(m + n \log n) \log M)$, slightly improving Theorem 3.3. (The correctness of EK capacity scaling follows from Lemma 3.5, applied one edge or vertex at a time, i.e., $I = 1$).

Next consider the *transportation problem with cost functions*. This problem allows parallel copies of an edge to have different costs. Specifically the cost of the p^{th} copy of an edge e, $1 \leq p \leq u(e)$, is given by $c(e,p)$, a nondecreasing function of p that can be evaluated in $O(1)$ time. As usual these costs are in $[-N..N]$, and each vertex v has a desired degree $u(v)$. The problem is to find a minimum cost perfect DCS for these degree constraints. Note that the desired DCS can still be represented by an integral function on the edges $d(e)$, where $0 \leq d(e) \leq u(e)$, since without loss of generality the DCS contains the $d(e)$ copies of e with smallest cost.

As examples of this problem, $c(e,p) = \lfloor a_e p \rfloor + b_e$ is the original DCS problem for $a_e = 0$ and a simple example of diminishing returns to scale for $a_e > 0$. Alternatively $c(e,p)$ could be, say, a piecewise quadratic function; in this case evaluating $c(e,p)$ for arbitrary p would probably involve a binary search on the breakpoints. (Note that in the definition of the transportation problem with cost functions, the restriction to nondecreasing cost functions $c(e,p)$ is crucial: without it the problem is NP-hard [GJ, p.214]. Also note that the solution to the problem is a multigraph with integral multiplicities, by definition. This assumption of integrality is also crucial. This issue is discussed further after Theorem 3.5 below for network flows, where real-valued flows make sense).

The algorithm of Theorem 3.3 works correctly for the transportation problem with cost functions, if we treat parallel copies of an edge with different costs as different edges. In the time bound, however, the term m now counts each edge e according to the number of distinct costs $c(e,p)$. We will show how to extend the capacity scaling algorithm to the transportation problem with cost functions, preserving the time bound of Theorem 3.3.

First we modify the cost scaling algorithm to preserve the time bounds of Theorems 3.1-2. The derivation of those theorems remains valid for cost functions and gives the desired time bounds, provided all individual steps are implemented to run in essentially the same time as before. This means implementing Step 1 of the main routine and *scale_match* in time $O(m)$ (even though they modify every cost) and similarly for *match*. This can be done because of the following observation. The conditions for a DCS D to be 1-feasibile are equivalent to a system with only two inequalities per edge $e = vw$,

$$c(e,d(e)) \leq y(v) + y(w) \leq c(e,d(e)+1) + 1.$$

Further the only copies of e that can be eligible are D-edges costing $c(e,d(e))$ and non-D-edges costing $c(e,d(e)+1)+1$.

Step 1 of the main routine and *scale_match* do not explicitly modify edge costs. Instead *match* computes the cost of an edge when it is needed in $O(1)$ time using arithmetic. Specifically, the p^{th} cost for vw is

$$trunc(\overline{c}(vw,p)/2^{k-s}) - y_0(v) - y_0(w), \tag{5}$$

where *trunc* denotes integer truncation, $k = \lfloor \log(n+2)N \rfloor$ is the number of cost scales, s is the index of the current cost scale, and y_0 denotes duals at the start of scale s.

The *match* routine starts by initializing D to contain all edges costing less than -1. This is done by examining each edge and adding smallest cost copies to D until the cost reaches -1. The time is $O(m+U)$, which suffices for the bounds of Theorems 3.1-2.

In the depth-first search of Step 1, it is unnecessary to know the multiplicity of each eligible edge when the search begins. Rather, costs $c(e,d(e))$ and $c(e,d(e)+1)$ are used to determine which edges have at least one eligible copy. When the depth-first search finds an augmenting path P, the next cost for each edge $e \in P$ is used to see if there is another eligible copy of e (i.e., for $e \in P \cap D$, another copy of e is eligible if $c(e,d(e)-1) = c(e,d(e))$, and similarly for $e \in P - D$). Thus the time for the depth-first search is still $O(m)$ plus the total augmenting path length. It is obvious that the Hungarian search, given costs $c(e,d(e))$ and $c(e,d(e)+1)$, uses time $O(m)$. Thus the bounds of Theorems 3.1-2 apply.

Now we modify the capacity scaling algorithm of Theorem 3.3. The new version works by scaling the domain of the cost functions. The closeness lemma (Lemma 3.5) generalizes as follows. Let G be a multigraph with c functions and u values for which D is a minimum cost maximum cardinality DCS. Form u^+ by adding one to the u-values of an arbitrary set of vertices and edges. Form c^+ so that for each edge e and $p \in [0..u(e))$,

$$c(e,p+1) \geq c^+(e,p+1) \geq c(e,p). \tag{6}$$

(Here $c(e,0) = -\infty$). Let I be the number of vertices with an increased u-value plus the number of edges with an increased u-value or some decreased c-value (so that $I \leq m+n$). Let D^+ be a minimum cost maximum cardinality DCS for c^+ and u^+, chosen so that $|D^+ \oplus D|$ is minimum ($D^+ \oplus D$ has the obvious interpretation).

Lemma 3.6. *$D^+ \oplus D$ can be partitioned into at most I simple alternating paths and cycles.*

Proof. The argument is an expanded version of Lemma 3.5; we will give only the new material. The definition of *new edge* is expanded to include a type (*iii*) new edge e, defined to have $d^+(e) > d(e)$ and $c(e,d(e)+1) > c^+(e,d(e)+1)$, where only the $d(e)+1^{st}$ copy of e is new. (Note that $d^+(e) - d(e)$ may be larger than one).

The argument expands in the case that P is an even length alternating path or cycle not containing a new edge, and we must show that it has zero net cost (with respect to c^+ and D^+). The net cost of P with respect to c^+ and D^+ is nonnegative, by the choice of D^+. Hence it suffices to show that the net cost of P with respect to c^+ and D is nonnegative.

This follows from the choice of D if for every edge e whose p^{th} copy is in $P \cap D^+$, $c^+(e,p) \geq c(e,d(e)+1)$. We prove this inequality as follows. The copy of e is not new and $p \geq d(e)+1$. Now consider two cases: If $p = d(e)+1$ then $c^+(e,p) = c(e,d(e)+1)$, as desired. If $p > d(e)+1$ then $c^+(e,p) \geq c(e,d(e)+1)$ by (6), as desired. ∎

This lemma justifies an algorithm similar to the capacity scaling algorithm. The main differences are as follows. Step 1, in addition to scaling d and u, scales the cost function domain. Specifically let c_0 denote the given cost function. Then for each e and $p \in [1..u(e)]$ (where $u(e)$ is the new u-value) Step 1 sets $c(e,p) \leftarrow c_0(e,\lfloor 2^{k-s}p \rfloor)$, where $k = \lfloor \log M \rfloor + 1$ is the number of capacity scales and s is the index of the current capacity scale. Observe that the DCS for the new d is a minimum cost maximum cardinality DCS for (the new) u-values rounded down to even numbers and (the new) $c(e,2p-1)$ increased to $c(e,2p)$. So Lemma 3.6 applies and justifies the

remaining steps: Step 2 defines G' as before but with costs changed in the obvious way to take cost functions into account. Step 3 computes D' using the cost scaling algorithm described above.

For efficiency, these three steps are not done explicitly. (For instance, doing Step 2 explicitly would use $\Theta(m^2)$ time, since an edge can be in G' with multiplicity $m+n$). Step 1 computes only two new costs for each edge e, $c(e,d(e))$ (already known) and $c(e,d(e)+1)$. To do Step 2, G' is initialized to contain only the cheapest copy of each edge of type v_1w_1, v_2w_2. This is the copy that will be added to the DCS D' first. Each copy comes from a cost computed in Step 1. When the cost scaling algorithm checks to see if there is another eligible copy of an edge e (in the depth-first search), the next higher (or lower) cost copy of e is computed, its cost is scaled down using (5) and it is used in the cost scaling algorithm.

Theorem 3.4. The transportation problem (capacitated or not) with cost functions can be solved in $O(nm \log n \log(nN) \log M)$ time. The space is $O(m)$. ∎

3.3. Network flow.

Our results extend to integral network flows. It is convenient to work with the problem of finding a minimum cost circulation, defined as follows [L]. Let G be a directed graph with nonnegative integral capacities $u(v)$ for each vertex v, and for each edge e, nonnegative integral capacities $u(e)$, lower bounds $\ell(e)$ and costs $c(e)$. The *minimum circulation problem* is to find a feasible circulation with smallest possible cost. (If vertex capacities are not given, setting $u(v) = \sum_{vw} u(vw)$ does not change the problem. The circulation problem includes the minimum cost flow problem as a special case. As already mentioned, the usual definition of the circulation (network flow) problem allows real-valued parameters. However note that if all capacities and lower bounds are integral, an optimum circulation (flow) that is integer-valued always exists [L]).

A minimum circulation problem on a network G can be transformed to a minimum perfect DCS problem on a bipartite multigraph B, as follows. A vertex $v \in V(G)$ corresponds to v_1, $v_2 \in V(B)$; B has an edge v_1v_2 of cost 0 and multiplicity $u(v)$. An edge $vw \in E(G)$ corresponds to $v_1w_2 \in E(B)$ with cost $c(vw)$ and multiplicity $u(vw) - \ell(vw)$. The degree constraints on B are

$$u(v_1) = u(v) - \sum\{\ell(vw) | vw \in E(G)\},$$
$$u(v_2) = u(v) - \sum\{\ell(wv) | wv \in E(G)\}.$$

A circulation on G corresponds to a perfect DCS on B costing less by exactly $\sum\{\ell(e)c(e) | e \in E(G)\}$. Thus the flow problem can be solved using the DCS algorithms given

above. Note that B has n vertices in each vertex set, $O(m)$ edges, $U = O(\sum\{u(v)|v \in V(G)\})$ and $\overline{m} = O(U + \sum\{u(e)|e \in E(G)\})$. In part (a) below, $\overline{m}$ is the number of edges, with each edge counted according to its capacity.

Theorem 3.5. A minimum cost circulation on a network with all edge capacities and lower bounds in $[0..M]$ can be found in the following time bounds (and space $O(m)$):

(a) $O(\min\{\sqrt{\overline{m}}, n^{2/3}M^{1/3}\}\overline{m}\log(nN))$.

(b) $O((\min\{\sqrt{mM}, n^{2/3}M^{1/3}, n\}m + \min\{mM\log(mM), n^2\sqrt{\overline{m}}\})\log(nN))$.

(c) $O(nm\log n\log(nN)\log M)$.

These bounds also hold when each edge cost is a convex function of its flow.

Proof. These bounds follow essentially from Theorems 3.1-4. Note that M does not necessarily bound the multiplicities in B, since we assume no bound on vertex capacities in G. Nonetheless the bound for part (b) holds: To show this use Corollary 3.1, with matching X containing all edges of the form v_1v_2; note that $M_X = M$. Also the bound for part (c) holds: There are $\log(mM)$ capacity scales, but the time bound involves the factor $\log M$ because the first $\log m$ scales are trivial. ∎

Note that in Theorem 3.5 (c), the algorithm for convex cost functions finds an optimal integral-valued flow. However this flow need not be the global optimum, which may involve real-valued flow values. Finding this solution appears to be much harder. For instance if the cost of an edge is a quadratic function of its flow, finding a minimum cost flow is NP-hard [GJ, H].

Next consider a minimum circulation problem in which $O(n)$ edges have finite capacity (every edge still has a lower bound, perhaps zero). Such problems arise as covering problems; a common special case is circulations with lower bounds but no upper bounds (e.g., the aircraft scheduling problem of [L, p.139]).

Theorem 3.6. A minimum circulation on a network with lower bounds but only $O(n)$ finite capacities, all in $[0..M]$, can be found in $O(n(m + n\log n)\log(nM))$ time and $O(m)$ space.

Proof. Without loss of generality assume that no cycle has negative cost and infinite capacity. Then it is easy to see that all infinite capacities (on edges or vertices) can be replaced by any number that is at least $S = \sum\{$If $c(e)$ is finite then $c(e)$ else $\ell(e)|e \in E(G)\} + \sum\{c(v)|v \in V(G)$, $c(v)$ is finite$\}$.

The algorithm is as follows: Find S and $k = \lceil\log S\rceil$. For each infinite capacity vertex v, set its capacity to S; for each infinite capacity edge e, set its capacity to $\ell(e) + 2^{k+1}$. Transform the new

circulation problem to a DCS problem, as above, and solve the DCS problem using EK capacity scaling.

The correctness of this algorithm follows from the opening remark. To estimate the efficiency, note that in the DCS problem, every infinite capacity edge of G has multiplicity 2^{k+1} and every vertex v_1, v_2 has degree constraint at most $S \leq 2^k$. Hence the first scale is trivial— no edges are in the DCS, and the duals can be set to any values small enough so that they are dominated on all edges. Every scale after the first has $I = O(n)$ (recall that I is the number of increased u-values), since the u-values of infinite capacity edges double. Hence the total time is $O(n(m+n\log n)\log S)$, implying the desired bound. ∎

Corollary 3.3. The directed Chinese postman problem can be solved in $O(nm\log n)$ time and $O(m)$ space.

Proof. We use the notation for the postman problem given in the Introduction. Transform the postman problem to a circulation problem as follows: Add new vertices s and t. For each start vertex v add edge sv with capacity $indegree(v) - outdegree(v)$; for each end v add edge vt with capacity $outdegree(v) - indegree(v)$. If q is the value of a maximum flow in this graph, add edge ts with lower bound q. (q is the cardinality of the desired set of paths P). Define the capacity of each vertex and each original edge of G as infinite. The problem is to find a minimum cost circulation from s to t.

The correctness of this transformation is obvious. To estimate the efficiency, first observe that q can be found in $O(nm)$ time (in any network with integer capacities and maximum flow value $O(m)$, a maximum flow can be found in $O(nm)$ time [G85a]).

The circulation problem can be solved in time $O(n(m+n\log n)\log n)$, by Theorem 3.6. For $m \geq n\log n$ this is the desired bound. If $m \leq n\log n$ the desired time is achieved by any algorithm that does $O(m)$ Hungarian searches. For instance the desired circulation can be found without scaling, by repeatedly finding a minimum cost augmenting path from s to t (the number of augmentations is at most m). Alternatively the DCS graph for Theorem 3.6 can be modified slightly so that the scaling algorithm does at most m augmentations, thereby achieving the desired bound in this case too. (The idea is to define capacities so that the scaling algorithm does at most one search per new unit of vertex capacity. Specifically, for a start vertex v, in the DCS graph define $u(v_1) = 2^k + indegree(v) - outdegree(v)$, $u(v_2) = 2^k$; similarly for an end vertex or other vertex). ∎

4. Conclusions.

Table I shows that in terms of asymptotic estimates, many network problems can be solved efficiently by scaling. Scaling algorithms also tend to be simple to program. For instance the assignment algorithm consists of an outer scaling loop plus an inner loop that does a depth-first search followed by a Dijkstra calculation. We believe that such algorithms will run efficiently in practice. Note that in the experiments done by Bateson [Ba] the scaling algorithm of [G85a] beat the Hungarian algorithm as long as the cost of the matching could be stored in a machine integer. Our assignment algorithm has even simpler code than [G85a] and so should do even better.

The assignment algorithm has a processor-efficient parallel implementation. Details are given in [GabT87b]. Also, the assignment algorithm extends to matching on general graphs. The time bound for finding a minimum perfect matching on a general graph is $O(\sqrt{n\alpha(m,n)\log n}m\log(nN))$. The algorithm is more complicated because of "blossoms" that occur in general matching, which compound the error due to scaling. Details are in [GabT87a].

Acknowledgments.

We thank Andrew Goldberg for sharing his ideas, which inspired this work.

References.

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[Ba] C.A. Bateson, "Performance comparison of two algorithms for weighted bipartite matching", M.S. Thesis, Department of Comp. Sci., University of Colorado, Boulder, Co. 1985.

[Bel] R.E. Bellman, "On a routing problem," *Quart. Appl. Math 16*, 1958, pp. 87-90.

[Ber86] D.P. Bertsekas, "Distributed asynchronous relaxation methods for linear network flow problems", LIDS Report P-1606, M.I.T., Cambridge, Mass.,1986; preliminary version in *Proc. 25th IEEE Conf. on Decision and Control*, December 1986.

[Ber87] D.P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem", LIDS Report P-1653, M.I.T., Cambridge, Mass., 1987.

[D] R.B. Dial, "Algorithm 360: Shortest path forest with topological ordering", *C. ACM 12*, 1969, pp. 632-3.

[EJ] J. Edmonds and E.L. Johnson, "Matching, Euler tours and the Chinese Postman", *Math. Programming 5*, 1973, pp. 88-124.

[EK] J. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM 19*, 2, 1972, pp. 248-264.

[ET] S. Even and R.E. Tarjan, "Network flow and testing graph connectivity", *SIAM J. Comput. 4*, 4, 1975, pp. 507-518.

[FT] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *Proc. 25th Annual Symp. on Found. of Comp. Sci.*, 1984, pp.338-346.

[G85a] H.N. Gabow, "Scaling algorithms for network problems", *J. of Comp. and Sys. Sciences 31*, 2, 1985, pp. 148-168.

[G85b] H.N. Gabow, "A scaling algorithm for weighted matching on general graphs", *Proc. 26th Annual Symp. on Foundations of Comp. Sci.*, 1985, pp. 90-100.

[G87] H.N. Gabow, "Duality and parallel algorithms for graph matching", manuscript.

[GabT87a] H.N. Gabow and R.E. Tarjan, "Faster scaling algorithms for general graph matching problems", manuscript.

[GabT87b] H.N. Gabow and R.E. Tarjan, "Improved parallel and sequential algorithms for network flow problems", manuscript.

[GalT] Z. Galil and E. Tardos, "An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm", *Proc. 27th Annual Symp. on Found. of Comp. Sci.*, 1986, pp.1-9.

[GJ] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, Ca., 1979.

[Go] A.V. Goldberg, "Efficient graph algorithms for sequential and parallel computers", Ph. D. Dissertation, Dept. of Electrical Eng. and Comp. Sci., MIT, Technical Rept. MIT/LCS/TR-374, Cambridge, Mass., 1987.

[GoT86] A.V. Goldberg and R.E. Tarjan, "A new approach to the maximum flow problem", *Proc.* 18th *Annual ACM Symp. on Th. of Computing*, 1986, pp. 136-146.

[GoT87a] A.V. Goldberg and R.E. Tarjan, "Solving minimum-cost flow problems by successive approximation", *Proc.* 19th *Annual ACM Symp. on Th. of Computing*, 1987, pp. 7-18.

[GoT87b] A.V. Goldberg and R.E. Tarjan, "Finding minimum-cost circulations by successive approximation", Technical Rept. CS-TR-106-87, Department of Comp. Sci., Princeton University, Princeton, New Jersey, 1987.

[H] P.P. Herrmann, "On reducibility among combinatorial problems," Report No. TR-113, Project MAC, MIT, Cambridge, Mass., 1973.

[HK] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM J. Comp. 2*, 4, 1973, pp. 225-231.

[K55] H.W. Kuhn, "The Hungarian method for the assignment problem", *Naval Res. Logist. Quart. 2*, 1955, pp. 83-97.

[K56] H.W. Kuhn, "Variants of the Hungarian method for assignment problems", *Naval Res Logistics Quart., 3*, 1956, pp. 253-258.

[L] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[Le] D. Lee, private communication, 1987.

[PS] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[T83] R.E.Tarjan, *Data Structures and Network Algorithms*, SIAM Monograph, Philadelphia, Pa., 1983.

[W] R.A. Wagner, "A shortest path algorithm for edge-sparse graphs", *J. ACM 23*, 1, 1976, pp. 50-57.

Strong Polynomial Bound	New Scaling Bound
Assignment problem	
$O(n(m + n\log n))$ [FT]	$O(\sqrt{n}m\log(nN))$
Shortest paths (single-source, directed graph, possibly negative lengths)	
$O(nm)$ [Bel]	$O(\sqrt{n}m\log(nN))$
Minimum cost degree-constrained subgraph of a bipartite multigraph	
$O(U(m + n\log n))$ [FT,G83]	$O(\min\{\sqrt{U}, n^{2/3}M^{1/3}\}\overline{m}\log(nN))$
Transportation problem (uncapacitated or capacitated)	
$O(\min\{U, n\log U\}(m + n\log n))$ [FT, EK, L]	$O((\min\{\sqrt{U}, n\}m + U\log U)\log(nN))$
Minimum cost flow	
$O(n^2(m + n\log n)\log n)$ [GalT]	$O(nm\log n\log(nN)\log M)$ convex cost functions allowed $O(n(m + n\log n)\log(nM))$ lower bounds only
Directed Chinese postman problem (with edge lengths)	
$O(m(m + n\log n))$ [FT, EJ, Le]	$O(nm\log n)$

Table I. Bounds for network problems.

Parameters:

n = number of vertices	U = total degree constraints
m = number of edges	N = maximum cost magnitude
$\overline{m}$ = number of edges counting multiplicities	M = maximum flow capacity or lower bound, or edge multiplicity

END
DATE
FILMED
7-88
DTIC